JAVA PROGRAMMING

Don C Josson

Department of Computer Application

- **Writing Javadoc Comments**
- In general, Javadoc comments are any multi-line comments ("/** ... */") that are placed before class, field, or method declarations. They must begin with a slash and two stars, and they can include special tags to describe characteristics like method parameters or return values. The HTML files generated by Javadoc will describe each field and method of a class, using the Javadoc comments in the source code itself. Examples of different Javadoc comments are listed below.
- **Simple Comments.** Normal Javadoc comments can be placed before any class, field, or method declaration to describe its intent or characteristics. For example, the following simple Student class has several Javadoc comments.
- /** … * Represents a student enrolled in the school. * A student can be enrolled in many courses. */ public class Student { /** * The first and last name of this student. */ private String name; /** * Creates a new Student with the given name. * The name should include both first and * last name. */ public Student(String name) { this.name = name; } } **Using Tags.** Tags can be used at the end of each Javadoc comment to provide more structured information about the code being described. For example, most Javadoc comments for methods include "@param" and "@return" tags when applicable, to describe the method's parameters and return value. The "@param" tag should be followed by the parameter's name, and then a description of that parameter. The "@return" tag is followed simply by a description of the return value. Examples of these tags are given below.
- /** * Gets the first and last name of this Student. * @return this Student's name. */ public String getName() { return name; } /** * Changes the name of this Student. * This may involve a lengthy legal process. * @param newName This Student's new name. * Should include both first * and last name. */ public void setName(String newName) { name = newName; } Other common tags include "@throws e" (to describe some Exception "e" which is thrown by a method) and "@see #foo" (to provide a link to a field or method named "foo").

- **Date class in Java (With Examples)**
- The class Date represents a specific instant in time, with millisecond precision. The Date class of java.util package implements Serializable, Cloneable and Comparable interface. It provides constructors and methods to deal with date and time with java.
- **Constructors**
- **Date()** : Creates date object representing current date and time.
- **Date(long milliseconds)** : Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.
- **Date(int year, int month, int date)**
- **Date(int year, int month, int date, int hrs, int min)**
- **Date(int year, int month, int date, int hrs, int min, int sec)**
- **Date(String s)**
- **Note :** The last 4 constructors of the Date class are Deprecated.

```
// Java program to demonstrate constuctors of Date
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Date d1 = new Date();
        System.out.println("Current date is " + d1);
        Date d2 = new Date(2323223232L);
        System.out.println("Date represented is "+ d2 );
    }
}
```

Output:

Current date is Tue Jul 12 18:35:37 IST 2016 Date represented is Wed Jan 28 02:50:23 IST 1970

- **boolean after(Date date) :** Tests if current date is after the given date.
- **boolean before(Date date) :** Tests if current date is before the given date.
- **int compareTo(Date date) :** Compares current date with given date. Returns 0 if the argument Date is equal to the Date; a value less than 0 if the Date is before the Date argument; and a value greater than 0 if the Date is after the Date argument.
- **long getTime()** : Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.
- **void setTime(long time)** : Changes the current date and time to given time.